(19) KOREAN INTELLECTUAL PROPERTY OFFICE

KOREAN PATENT ABSTRACTS

(11)Publication number: 000028684 A
(43)Date of publication of application: 25.05.2000

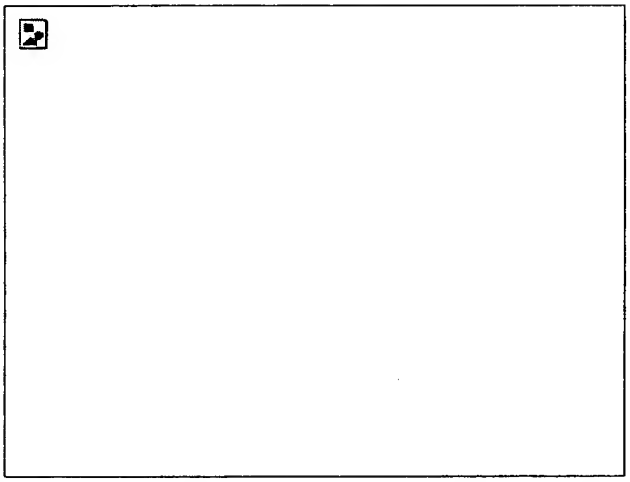
(21)Application number: 990039630
(22)Date of filing: 15.09.1999
(30)Priority: 19.10.1998 EP 98 98119826

(51)Int. Cl G06F 9/00

(71)Applicant: INTERNATIONAL BUSINESS MACHINES CORPORATION
(72)Inventor: BINANT ANDRE
GAMMA ERICH
MED HABANSAN J.

(54) IMPROVED PRESENTATION METHOD AND SYSTEM FOR COMMUNICATION BETWEEN USER STATION AND APPLICATION PROGRAM

(57) Abstract:



PURPOSE: A system is provided to cooperate with an application program existed in a server all the time for being displayed after being inputted in an interactive method with a graphic user interface performed in a user station.

CONSTITUTION: An interactive cooperation is performed through a link or a network. The structure of a presentation element and a management mechanism are established in a user station. Herein, the structure of presentation element maintains and delivers dialogue data, and the management mechanism realizes and controls the presentation element. The structure corresponding to a counterpart element, which maintains and delivers the dialogue data, is established by an application program. Two corresponding data sets for each dialogue element are maintained for the dialogue typed exchange of dialogue data in the corresponding presentation elements and in the corresponding counterpart elements. Herein, the user station and the application program cooperate with the presentation elements and the counterpart elements regardless of a transmission through the link or the network.

Legal Status
1. Appliaction for a patent (19990915)
2. Decision on a registration (20020222)

Processing

(19) 대한민국특허청(KR)
(12) 공개특허공보(A)

(51) Int. Cl.[7]
G06F 9/00

(11) 공개번호 특2000-0028684
(43) 공개일자 2000년05월25일

(21) 출원번호 10-1999-0039630
(22) 출원일자 1999년09월15일
(30) 우선권주장 98119826.0 1998년10월19일 EPO(EP)
(71) 출원인 인터내셔널 비지네스 머신즈 코포레이션
미국 10504 뉴욕주 아몬크
(72) 발명자 감마에리히
스위스CH-8604폴케츠빌토낙커슈트라세5
매드하반산제이
스위스CH-8052취리히샤프하우저슈트라세452
바이난트안드레
스위스CH-8006취리히블루엠리잘프슈트라세19
(74) 대리인 원석희, 박해천

심사청구 : 있음

(54) 사용자 스테이션과 애플리케이션 프로그램 사이의 통신을위한 개선된 프리젠테이션 방법 및 시스템

요약

본 발명은 사용자 스테이션과 서버와 같은 중앙 위치에 상주하는 공통적으로 이용되는 애플리케이션 프로그램의 대화식 협동을 위한 시스템 및 방법에 관한 것으로서, 대화식 협동은 링크 또는 네트워크를 통해 이루어진다. 본 발명에 따르면, 대화 데이터를 전달하고 유지하기 위한 프리젠테이션 요소의 구조와 이 프리젠테이션 요소를 구현하고 제어하기 위한 관리 메카니즘이 사용자 스테이션에 설정된다. 대화 데이터를 전달하고 유지하기 위한 카운터파트 요소의 대응하는 구조가 애플리케이션 프로그램에 의해 설정되며, 관련 프리젠테이션 요소 및 대응하는 카운터파트 요소에서 각각 대화 요소를 위한 2개의 대응하는 데이터 셋트가 대화 데이터의 대화식 교환을 위해 유지된다. 이것은 사용자 스테이션과 애플리케이션 프로그램이 각각 링크 또는 네트워크를 통한 전송과 무관하게, 프리젠테이션 요소 및 카운터파트 요소와 협동할 수 있는 방식으로 수행된다.

대표도

도1

색인어

사용자 스테이션, 애플리케이션 프로그램, 대화식 협동, 대화 데이터, 프리젠테이션 요소, 카운터파트 요소

명세서

도면의 간단한 설명

도1은 본 발명을 이용한 시스템의 블록도.

도2는 도1의 시스템에서의 프리젠테이션 요소 및 카운터파트 요소(위지트/프룩시)의 페어/스플릿 특성을 예시한 도면.

도3은 특정 애플리케이션(Dossier)에 사용되는 프리젠테이션 요소(위지트)의 구조를 도시한 도면.

도4는 프리젠테이션 및 카운터파트 요소 각각이 모델들 사이에서의 중간 저장 및 직접 전송을 위한 각각의 데이터 구조의 관련 모델을 구비하고 있는 본 발명의 특정 실시예를 도시한 도면.

도5는 링크 양쪽에서의 대칭 입출력 및 버퍼링 메카니즘을 이용한 프리젠테이션 요소와 카운터파트 요소 사이의 통신 구성도.

도6a 및 도6b는 사용자 스테이션의 사용자 인터페이스 엔진이 대화 세션을 개시하기 위해 공통(중앙) 애플리케이션 서버의 애플리케이션 제어기와 협동할 수 있는 방법을 도시한 도면.

도7은 본 발명을 이용한 시스템에서의 테이블 데이터의 대화식 전송을 위한 상세도.

도8은 위지트의 구현에 대해 설명한 파이 차트.

* 도면의 주요 부분에 대한 부호의 설명

11:대화 요소	12:디스플레이
13:사용자 스테이션	14:서버
15:위지트	16:프록시
17:UI 엔진	18:애플리케이션
19:통신 링크	20:애플리케이션 프로그램
25:데이터베이스	

발명의 상세한 설명

발명의 목적

발명이 속하는 기술 및 그 분야의 종래기술

본 발명은 데이터가 중앙 위치, 예를 들어, 서버에 상주하는 공통적으로 사용되는 애플리케이션 프로그램과 협동하여, 사용자 스테이션에서 실행되는 그래픽 사용자 인터페이스와 대화식으로 입력되어 디스플레이될 수 있는 시스템에 관한 것이다.

공통(common) 서버내의 애플리케이션 프로그램과 원격 사용자 스테이션 사이의 대화식 협동(interactive coorperation)을 위한 현재 사용되는 시스템은 범용 대화식 세션(versatile interactive sessions)을 허용할 수 있다. 그러나, 이들중 대부분은 사용될 애플리케이션에 특히 적합한 많은 처리 기능을 요하는 "중량급(heavy)" 사용자 스테이션을 필요로 하며, 이것은 결과적으로 값비싼 사용자 스테이션을 초래하며, 특히, 애플리케이션 프로그램이 수정되거나 개선되는 경우에 매우 많은 수의 사용자 스테이션을 갱신하기 위한 막대한 노력을 필요로 한다. 현재 사용중에 있는 다른 시스템들은 간단한 사용자 스테이션을 갖고 있지만, 매번의 대화식 입력 및 출력 연산에 수반되어야 하는 애플리케이션의 호스트가 되는 중앙 서버와 사용자 스테이션 사이에서 여러번의 전송을 필요로 한다. 또한, 현재의 시스템의 성능은 링크 품질에 매우 의존적이며, 따라서, 높은 대역폭 및 낮은 대기시간이 요구된다.

발명이 이루고자하는 기술적 과제

본 발명의 목적은 사용자 스테이션에서 애플리케이션-특유의 프로그램을 필요로 하지 않으면서, 사용자 스테이션에서 프리젠테이션 로직의 일부 태스크(입출력 데이터의 준비 및 전송)를 처리할 수 있도록 하는, 중앙 위치의 애플리케이션 프로그램과 사용자 스테이션 사이의 대화식 협동을 위한 시스템 및 방법을 제공하는 것이다.

본 발명의 다른 목적은 이러한 시스템에서 애플리케이션 프로그램과 사용자 스테이션 사이에서 전송되어야 하는 데이터 및 메시지의 양을 감소시키고, 대화 세션 동안에 데이터 전송을 위해 링크를 최적으로 이용할 수 있도록 하는 것이다.

본 발명의 또다른 목적은 이러한 시스템에서 매우 많은 양의 데이터 전송이나 또는 제한된 링크(네트워크) 기능에 의해 유발될 수도 있는 과도한 지연을 없애는 것이다.

이들 목적은 링크 또는 네트워크를 통한 서버내의 애플리케이션 프로그램과 사용자 스테이션 사이의 대화식 협동을 위한 본 발명의 방식에 의해 달성된다. 본 발명에 따르면, 애플리케이션 프로그램과 사용자 스테이션 사이에 제공되는 프리젠테이션 계층(presentation layer)이 있다. 이 프리젠테이션 계층은 부분적으로는 사용자 스테이션에서 구현되고, 부분적으로는 서버에서 구현된다. 이것은 사용자 스테이션에 디스플레이되는 대화 요소(dialog element)를 나타내는 한쌍의 프리젠테이션 요소를 포함한다. 이 한쌍중에서 하나의 프리젠테이션 요소는 사용자 스테이션에 제공되고, 상기 한쌍중에서 대응하는 인터페이스 요소는 서버에 제공된다. 또한, 사용자 스테이션에서 실행되며, 제시된 대화 요소를 제어하는 프로그램이 제공된다.

링크 또는 네트워크를 통한 애플리케이션 프로그램과 원격 사용자 스테이션 사이의 대화식 협동은 본 발명의 방식에 의해 개선된다. 이 방식에 따르면, 대화 데이터를 전송하고 유지하기 위한 프리젠테이션 요소의 구조와 이 프리젠테이션 요소를 구현하고 제어하기 위한 관리 메카니즘이 사용자 스테이션에 설정된다. 대화 데이터를 전송하고 유지하기 위한 카운터파트 요소의 대응하는 구조가 애플리케이션 프로그램에 의해 설정된다. 이것은 사용자 스테이션과 애플리케이션 프로그램이 각각 링크 또는 네트워크를 통한 전송에 관계없이 프리젠테이션 요소 및 카운터파트 요소와 각각 협동할 수 있는 방식으로 수행된다.

발명의 구성 및 작용

용어 정의

"사용자 스테이션"은 입력 및/또는 출력 데이터의 준비 및 전송을 할 수 있도록 하는 시스템이며, 즉, 사용자 스테이션은 사용자가 정보를 대화식으로 입력하고 및/또는 수신할 수 있는 시스템이다. 예를 들어, 워크스테이션, 단말, 네트워크 컴퓨터, 개인용 컴퓨터, 데스크탑 머신, 휴대용 시스템, 핸드헬드 컴퓨팅 장치 등이 있다. 사용자 스테이션은 여기서, 그 사용자 스테이션을 중앙 위치 또는 서버로 불리는 다른 시스템에 접속하는 링크 또는 네트워크가 있는 경우에 원격 사용자 스테이션으로 언급한다. "원격(remote)"이란 용어는 사용자 스테이션이 중앙 위치 또는 서버로부터 멀리 떨어져 있다는 것을 나

타내기 위해 사용된 것이 아니다. 원격 사용자 스테이션은 바로 인접하여 위치될 수도 있다.

"애플리케이션 프로그램"은 데이터를 처리하는 코드이다. 이것은 소프트웨어 패키지의 일부 또는 모듈이 될 수 있으며, 또는 전체 패키지가 될 수 있다. 애플리케이션 프로그램의 통상적인 예로는, 텍스트 및 이미지 처리, 스프레드 시트(spread sheets), 미팅 캘린더 등을 위한 오피스 환경에 사용되는 프로그램이나, 또는 보험사, 은행, 제조 설비, 호텔, 콜 센터 등을 위한 특정 애플리케이션이 있다. 다른 예로는, 통신 네트워크를 구성하고 및/또는 모니터하기 위해 사용되는 네트워크 관리 소프트웨어가 있다. 본 발명은 대화식 시스템을 구축할 수 있도록 한다.

"중앙 위치(central location)"는 애플리케이션을 호스트할 수 있는 서버(애플리케이션 서버로도 불림)나 또는 어떤 다른 설비를 말한다. 통상적으로, 애플리케이션은 소프트웨어의 형태로 호스트된다. 서버의 예로는, 워크스테이션, 개인용 컴퓨터, 데스크탑 머신, 휴대용 컴퓨터, 서버 머신, 메인프레임 컴퓨터 등이 있다.

"링크" 또는 "네트워크"는 데이터 및/또는 정보의 교환을 허용하는, 사용자 스테이션과 중앙 위치 사이의 접속을 말한다. 이것은 유선 또는 무선이 될 수 있으며, 또는 이들의 조합이 될 수 있다.

모든 다른 용어 및 표현은 도입될 때 설명되거나 또는 본 명세서의 문맥상으로 보아 이 기술분야에 통상의 지식을 가진자에게 자명한 것이다.

1) 환경

본 발명의 환경은 애플리케이션 프로그램을 포함하고 있는 데이터 처리 설비에 적어도 하나의 사용자 스테이션이 접속되어 있는 시스템이다. 복수-사용자(multi-user) 환경에는 애플리케이션 프로그램을 포함하는 공통 데이터 처리 설비에 접속된 복수의 사용자 스테이션이 존재한다. 사용자는 예를 들어, 고객 계정을 매일 갱신하는 것과 같이, 그가 애플리케이션을 실행하고 있는 그의 스테이션에서 대화식으로 정보를 입력하고 수신할 수 있다.

사용자 스테이션과 공통 설비 사이에서 데이터 처리 및 저장을 분산시키기 위한 다양한 가능성이 있다. 한가지 극단적인 예는 사용자에게는 단지 데이터를 디스플레이하고 입력을 수신하는 단순한 단말을 제공하고, 반면에 모든 처리는 중앙 컴퓨터에서 애플리케이션에 의해 수행되도록 하는 것이다. 이것은 네트워크를 통한 많은 전송량을 필요로 한다. 다른 부정적인 결과는 단말에 국부적 지능(local intelligence)이 없는 경우에 사용자 인터페이스의 품질이 낮아진다는 것이다. 다른 극단적인 예는 애플리케이션 프로그램이 모든 처리를 수행할 수 있는 강력한 사용자 스테이션을 제공하고, 단지 대화 세션의 결과만이 중앙 데이터베이스에 저장되도록 하는 것이다. 이 경우에, 사용될 애플리케이션에서의 변경 또는 개선은 모든 사용자 스테이션의 구조 및 프로그램의 갱신을 필요로 하며, 이것은 빈번한 개선 및 갱신에 방해가 된다.

본 발명은 다양한 애플리케이션에 대해 또한 애플리케이션을 변경하기 위해 사용자 스테이션이 보편적으로 사용될 수 있고 네트워크를 통한 전송이 최소로 감소될 수 있는 방식으로 사용자 스테이션과 공통 설비(예, 서버) 사이의 대화식 협동을 위한 처리 및 저장, 즉 프리젠테이션 로직이 분산되는 시스템을 제공한다.

도1은 본 발명을 포함하는 시스템의 블록도이다. 계속해서 본 발명의 기본 개념 및 세부사항에 대해 설명한다.

2) 본 발명의 기본적인 제안

사용자와 시스템 사이의 대화를 위해, 디스플레이(12) 상의 특정 필드에 있는 다양한(물리적인) 대화 요소(11)가 사용되며, 예를 들어, 데이터를 입력하기 위한 박스, 클릭을 위한 버튼, 명령 항목이 있는 메뉴바, 테이블 등이 있다. 이들 대화 요소(11)는 사용자에게 데이터 및 메시지를 가시적으로 출력하고 사용자에 의한 데이터 입력을 위해 사용된다.

본 발명은 대화 요소(11) 각각에 대해(또는 이러한 요소들의 조합된 그룹 각각에 대해) 사용자 스테이션(13)과 애플리케이션(18) 사이에 별도의 논리적인 경로를 제공한다. 각각의 대화 요소(11)에 대해, 각각의 논리적인 경로는 사용자 스테이션 측에서, 이후에 "프리젠테이션 요소" 또는 짧은 "위지트(widget)"로 불리는, 대화 요소(11)의 논리적 표기를 포함하고, 애플리케이션(공통 서버) 측에서, 이 논리적 표기의 카운터파트(16)를 포함하며, 이것은 다음에 "카운터파트 요소" 또는 "프록시(proxy)"로 불린다. 이 구조는 도1에 도시되어 있다.

따라서, 프리젠테이션 로직은 사용자 스테이션(13)과 서버(14) 사이에 분리되어 있다. 사용자 스테이션(13)은 단지 그것에 제공된 프리젠테이션 요소(15)(위지트)와만 대화하고(마치 그것이 애플리케이션(18)과 직접 대화하는 것처럼), 서버(14)내의 애플리케이션 프로그램은 카운터파트 요소(16)(프록시)와 대화한다. 이러한 분리는 사용자와 애플리케이션 프로그램에 투명하며, 즉, 이들은 마치 그들이 대화 요소(11)와 직접 통신하고 있었던 것처럼 동작한다. 데이터 및 메시지의 실제적인 전송은 사용자 또는 애플리케이션 프로그램의 참여없이 위지트(15)와 프록시(16) 사이에서 수행된다. 그러므로, 다시 말하면, 사용자 스테이션(13)의 디스플레이(12) 상의 대화 요소(11)와 서버(14) 상의 애플리케이션 프로그램 사이의 논리적인(프리젠테이션) 경로는 섹션으로 분리되며, 위지트(15)와 프록시(16)가 중간 "버퍼"를 구성한다.

본 발명은 매우 경량급의 사용자 스테이션(13)에 의한 애플리케이션(18)의 개발을 지원한다.

본 발명의 주요 아이디어는 중앙 제어 애플리케이션 서버(14) 상에서 애플리케이션(18)을 실행하는 것이다. 이것에 의해, 시스템 유지관리의 중앙집중화와 사용자 스테이션 관리의 비용 감소가 가능해진다. 애플리케이션(18)의 프리젠테이션 파트만이 사용자 스테이션(13)에서 실행된다. 이 프리젠테이션 파트는 보편적인 이용에 적합한 소형 프로그램에 의해 구현되며, 이 프로그램은 특히 프리젠테이션 요소(15) 또는 위지트를 구현(및 관리)한다.

그러므로, 이러한 솔루션의 중요한 구성요소는 사용자 스테이션(13) 상에 구현되는 이 인터페이스 프로그램이며, 이것은 다음에 "사용자 인터페이스 엔진(UI 엔진)"(17)으로 불린다. 이 UI 엔진(17)은 원격 사용자 스테이션(13) 상에서 애플리케이션(18)의 사용자 인터페이스를 실현하며, 이것은 최종 사용자와 대화한다.

UI 엔진(17)은 단지 프리젠테이션 로직만을 실행하며, (데이터 포맷 검증과 같은 기본적인 태스크를 제외하고는) 애플리케이션 로직을 실행하지 않는다. UI 엔진(17)과 애플리케이션(18) 사이의 실제적인 통신은 프리젠테이션 요소(15)(위지트)와 관련 카운터파트 요소(16)(프록시) 사이에서 수행된다. 애플리케이션(18)은 초기에, 사용자 인터페이스를 표현해야 하는 위지트(15)의 선택 및 구조에 관해 UI 엔진(17)에 명령한다. UI 엔진(17)은 이들 위지트(15)를 필요한 구조로 활성화시키고(activates), 사용자가 위지트(15)에 의해 표현된 대화 요소(11)와 대화할 때 및 애플리케이션(18)으로부터 데이터가 필요로 될 때, 애플리케이션(18)에 통지한다.

그러므로, 사용자 스테이션(13)에서 실행되는 것은 다양한 애플리케이션(18)으로부터 보편적으로 사용될 수 있는 UI 엔진(17)이 전부이다. 사용자 스테이션(13)에서 실행되는 애플리케이션 특유의 코드가 없기 때문에, 각각의 사용자 스테이션(13)의 애플리케이션 특유의 관리가 필요로 되지 않는다.

애플리케이션(18)은 고레벨 사용자 인터페이스 프로토콜을 통해 UI 엔진(17)과 통신한다. 이 프로토콜은 저대역폭 저속 통신 링크(19)로 축소(scale down)되도록 설계된다. 애플리케이션 개발자는 분산을 처리하는 특수한 툴키트(tool kit)에 의해 이 프로토콜로부터 보호될 수 있다. 다시 말하면, 애플리케이션 개발자는 그들이 분산형 애플리케이션을 개발하고 있다는 것을 인식하지 않아도 된다. 이러한 분산은 막후에서 처리된다. UI 엔진(17)은 입수가능한 웹 브라우저 내부에서 애플릿으로서 실행되거나 또는 외부의 헬퍼(helper) 애플리케이션으로서 실행될 수 있다.

HTML(HyperText Markup Language)에 기반한 전형적인 웹 애플리케이션과 대조적으로, 본 발명을 이용하는 시스템을 위한 애플리케이션(18)의 개발자는 동적 HTML 페이지 생성 및 CGI와 같은 서버 확장 메카니즘을 처리하지 않아도 된다. 이 애플리케이션(18)은 자바 및 스몰토크(Smalltalk)와 같은 통상적인 프로그래밍 언어를 이용하여 개발될 수 있다. 사용자 스테이션(13)이나 또는 서버(14) 어디에도 이종 기술(HTML, Plugins, Java, JavaScript)이 필요로 되지 않는다. 이것은 개발의 용이성 뿐만 아니라 견고성(robustness)도 개선시킨다. 또한, 사용자 인터페이스는 HTML에 의해 제한받지 않는다.

3) 위지트 및 구조의 세부사항

대화 세션을 위해 애플리케이션(18)을 구축하는 것은 특수한 객체 셋트(카운터파트 요소)(16)를 필요로 한다. 이들은 애플리케이션 프로그래밍 인터페이스 및 기능적 비헤이비어(behavior)를 갖으며, 이것은 "정상적인(normal)" 위지트와 다르지 않지만, 사용자 인터페이스를 전혀 갖고 있지 않다. 이들 카운터파트 요소(16) 각각은 통신 메카니즘에 의해 UI 엔진(17) 내의 대응하는 "리얼(real)" 위지트(15)와 대화할 수 있으며, 리얼 위지트(15)에 대한 프록시로서 작용한다. 이들 모든 위지트 구성은 2개의 절반 객체(half objects), 즉, 애플리케이션 프로그래밍 인터페이스 "카운터파트 요소"(16)(프록시)와 리얼 사용자 인터페이스 "프리젠테이션 요소"(15)(위지트)(도1 및 도2 참조)로 분할된다.

애플리케이션(18)은 request(요구), events(이벤트), 및 callbacks(콜백)으로 구성된 프로토콜을 통해 UI 엔진(17)과 통신한다. 사용자 대화는 통상적으로, 몇몇 저레벨 이벤트(예, "마우스 클릭")를 초래하며, 이것은 먼저 UI 엔진(17)에 의해 처리되고, 다음에 의미상의 이벤트(semantic event)("액션 실행")로 변환되며, 이 이벤트는 링크 또는 네트워크(18)를 통해 다시 애플리케이션(18)으로 전달된다. 이들 이벤트는 통상적으로, 위지트 구성의 다른 절반(프록시)을 동기시키고 다음에 소정의 애플리케이션 특유의 액션을 트리거시키기 위해 사용된다. 만일 UI 엔진(17)이 애플리케이션의 카운터파트 요소(16)(프록시)로부터 소정의 데이터를 필요로 하면, 그것은 콜백을 트리거시킨다.

4) 구조

위지트(15)(및 프록시(16))는 하이어라키(도3 참조)를 형성한다. 루트(root)에는, 애플리케이션(18)의 글로벌 상태(global state)를 조작하고(종단 등) 윈도우 또는 "쉘(shell)"(22)의 리스트를 유지하기 위한 방법을 제공하는 애플리케이션 위지트(21)가 있다. 쉘(22)은 선택적 메뉴바(23)(메뉴 항목을 가진 메뉴 트리(23))와 콘텐트(content) 영역을 구비한 탑-레벨(rop-level) 윈도우를 나타낸다. 콘텐트 영역은 합성(composite) 위지트의 트리이다. 합성 위지트는 트리의 내부 노드를 형성하고, 레이아웃을 구현한다. 단순한 위지트는 트리의 잎(leaves)에 해당한다. 도3은 "더지에이(Dossier)"로 불리는 샘플 애플리케이션으로부터의 트리의 일부분을 도시하고 있다. 다음에는 위지트 카테고리의 보다 상세한 설명이 제공된다.

절반 객체 스플릿(위지트/프록시)에 걸쳐 상태를 유지하는 것은 미묘한 문제이다. 누구나 통신 문제가 있거나 사용자 스테이션(13)이 고장남으로 인해 애플리케이션(18)을 사용불가능한 상태로 두는 것을 원하지 않는다. 그러므로, UI 엔진(17)은 "개념상 무상태(conceptually stateless)"이며, 모든 상태는 애플리케이션(18)에서 유지된다. 물론, 소정의 상태가 UI 엔진(17)에서 유지되지만(예, 위지트 하이어라키), 단지 캐시의 형태로서 유지된다. (예를 들어, UI 엔진(17)을 리셋함으로써) 캐시를 클리어시키고, 애플리케이션(18)으로부터의 모든 정보를 UI 엔진(17)으로 재전송하는 것은 항상 가능하다.

이러한 기능은 절반 객체(프록시/위지트) 사이의 통신 원리에 중요한 영향을 미친다. 통상적으로, 애플리케이션 측의 프록시(16)는 변경되며(상태 변경), 그러면, UI 엔진(17)의 관련 위지트(15)에서의 대응하는 변경을 위한 시도(위지트(15) "동기화")가 이루어진다. 만일 (통신이 타임아웃 되거나 UI 엔진(17)이 다운됨으로 인해) 관련 UI 엔진 위지트(15)가 없으면, "동기화" 단계는 무연산 명령(no-op)이 되지만, 프록시(16)는 일관된 상태에 있게 된다. 만일 UI 엔진(17)이 나중에 회복되면, 애플리케이션(18)은 그것에 재접속되고, 현재 상태로 "동기화(synchronizes)"된다.

본 발명에 따른 시스템은 저대역폭 접속(19)으로 축소되도록 설계된다. 그러므로, 네트워크 대기시간 및 네트워크 대역폭이 일부의 설계 치수에 영향을 준다. 절반 객체들(프록시/위지트) 사이의 통신은 최소화

되어야 하며, 요구는 사용자 인터페이스가 느려지는 것을 피하기 위해 단일 메시지로서 함께 일괄처리되어야 한다. 본 발명은 가시적이거나(visible)(예, 10000개의 로우를 가진 테이블에서 단지 10개의 가시적인 로우), 또는 곧 가시적이 될 가능성이 높은 (예, 상기 테이블에서 그 다음 10개의 로우) 프리젠테이션 데이터를 전송함으로써 통신 오버헤드를 최소화할 수 있도록 한다.

높은 대기시간 환경을 해결하기 위해, 시스템은 단일 요구들을 일괄처리(batch)해야 한다. 이러한 일괄처리는 요구들이 대부분 비동기인 경우, 즉, 요구들이 중간 응답을 필요로 하지 않고, 송신측(sender)을 블록킹하지 않는 경우에만 가능하다. 결과적으로, 애플리케이션(18)과 UI 엔진(17) 사이의 통신은 대부분 비동기이다. 예를 들어, 만일 UI 엔진(17)이 테이블을 드로잉해야 한다면, 그것은 그 테이블의 가시적인 부분에 대한 데이터를 요구하기 위해 다시 애플리케이션(18)으로 요구를 전송한다. UI 엔진(17)은 요구한 데이터를 즉시 입수하려고 대기하지 않고, 대신에 위치 홀더(place holder)를 드로잉한다. 데이터가 도달하길 대기하지 않기 때문에, UI 엔진(17)은 블록킹하지 않고, 응답상태를 유지한다. 네트워크 대기시간 및 애플리케이션 응답도(responsiveness)에 따라, 요구한 데이터는 나중에 비동기적으로 도달하게 되며, 위치 홀더 데이터를 대체하게 된다.

위지트 카테고리 개요

1. 자원

자원은 사용자 인터페이스 요소 자체는 아니지만, 이들 요소를 구성하기 위해 사용되며, 따라서, 애플리케이션(18)뿐만 아니라 UI 엔진(17)에서 실행되어야 하는 모든 객체들이다. 예를 들어, 폰트(fonts), 비트맵(bitmaps), 이미지(images), 커서(cursors) 등이 있다.

2. 위지트

위지트는 버튼(buttons), 레이블(labels), 편집가능 필드(editable fields), 메뉴(menus) 및 메뉴항목(menu items)과 같이 간단한 것에서부터 1차원 스크롤링 리스트 및 2차원 테이블 또는 트리 디스플레이와 같이 보다 복잡한 것까지 모든 종류의 프리젠테이션 요소들이다.

3. 레이아웃

이 카테고리내의 위지트는 그 차일드들(children)에 대한 특정 레이아웃 정책(policy)을 구현하는 합성 위지트이다.

4. 쉘(shells)

쉘(22)은 모든 위지트 트리의 루트를 형성하는 탑레벨 위지트이다. 쉘(22)은 통상적으로 양식(modal) 또는 비양식(non-modal) 윈도우로서 표현되며, 선택적으로는 메뉴바(23)를 갖고 있다. 쉘(22)은 이들 요소의 협조(collaboration)를 제어한다. 쉘(22)의 예로는 "표준" 쉘, 다이얼로그(dialogues), 앨러트(Alerts), 및 전체 애플리케이션을 관리하는 루트 쉘("Application")이 있다.

5. 모델

이 카테고리는 위지트 카테고리 2 이상의 위지트의 일부에 대한 데이터 소스로서 사용될 수 있는 클래스를 포함한다. 모델은 공지된 MVC(model view controller) 패러다임의 모델들과 동일하며, 위지트는 뷰 및 제어기 구성요소를 표현한다.

모델-기반 위지트

대부분의 위지트(카테고리2로부터)는 2개의 상이한 애플리케이션 프로그래밍 인터페이스를 지원한다. 제1 인터페이스는 모든 위지트가 내장(built-in) 모델을 갖고 있는 것으로 가정하고, 그 모델을 억세스하고 변경하기 위한 인터페이스를 제공한다. 예를 들어, 어떤 필드는 텍스트 모델과 절차(methods) setText 및 getText를 갖고 있다. 또한, 위지트는 예를 들어, 디스플레이를 위해 어떤 폰트를 사용하고 얼마나 많은 문자를 사용할 것인지와 같이, 사용자 인터페이스 요소의 가시적인 외관을 명시하기 위한 절차를 제공한다.

제2 애플리케이션 프로그래밍 인터페이스는 독립된 객체인 외부 모델(31)(도4 참조)을 이용한다. 서로 다른 위지트들(32)이 동일한 애플리케이션 데이터 구조(동일한 데이터 객체의 복수의 뷰)로부터 데이터를 수신하는 경우에, 이들은 동일한 모델(31)을 공유할 수 있다.

이러한 패러다임에 의하면, 애플리케이션(18)은 통상적으로 모델 프록시(31/34) 및 위지트/프록시(32/33)를 둘 다 생성하고 구성하며, 위지트/프록시(32/33)는 거의 이용하지 않는다. 애플리케이션(18)이 UI 엔진(17)에서 제시되는 데이터를 갱신해야할 때마다, 그것은 모델 프록시(31)와 대화한다. 애플리케이션(18)이 변경된 데이터를 검색하길 원하면, 그것은 그것에 대해 위지트/프록시(32/33)가 아니라 모델(31)에 질의한다.

이러한 애플리케이션 프로그래밍 인터페이스의 예로는, Table widget/proxy(테이블 위지트/프록시)와 TableModel(테이블 모델)이 있다(Table widget/proxy는 내장 모델을 갖고 있지 않으며, 모델-기반 애플리케이션 프로그래밍 인터페이스만을 지원한다). TableModel은 로우와 칼럼을 가진 2차원 데이터 구조를 구현한다. 모델의 UI 엔진 절반부는 Table 위지트의 데이터 억세스 요구가 즉시 충족될 수 있도록 캐시를 포함하고 있다. 만일 캐시가 유효 데이터를 포함하고 있지 않으면, TableModel은 위치 홀더를 반송하고, 애플리케이션 절반부 모델로부터 새로운 데이터를 비동기로 요구한다. 이 데이터가 도달하면, 종속(dependent) 위지트는 통지를 받고 갱신된다.

TableModel의 2개의 절반부 사이에 사용되는 갱신 정책은 구성가능하다. 동기화 폴리시는 전송되는 데이터의 양을 감소시킴으로써 저대역폭, 저대기시간 접속을 위한 통신을 최적화한다. 고대역폭, 저대기시간을 위한 다른 정책도 들어오고 나가는 요구의 수를 감소시키지만, 전송되는 데이터의 크기를 최소화하지는 못한다.

모델-기반 애플리케이션 프로그래밍 인터페이스의 다른 예로는 FormModel(폼 모델)이 있다. FormModel은 이종의 사전(heterogeneous dictionary)과 유사하게, 명명되고 분류된 애트리뷰트(named and typed attributes)들의 셋트를 표현한다. 카테고리2로부터의 대부분의 위지트/프록시는 FormModel 및 애트리뷰트 명칭을 이용하여 초기화될 수 있다. 스트링 애트리뷰트, 부울 애트리뷰트상의 체크박스(CheckBoxes) 및 목록 타입(enumeration type) 애트리뷰트상의 라디오버튼(RadioButtons)의 그룹에 필드들이 사용될 수 있다. 이들 위지트/프록시는 그 내장 모델을 무시하게 되지만, 지정된 FormModel의 명명된 애트리뷰트의 변경을 추적하고 갱신을 허용한다.

TableModel와 유사하게, 상이한 동기화 정책은 상이한 네트워크 특성을 수용할 수 있다. 한가지 정책은 모든 애트리뷰트 변경시마다 FormModel의 애플리케이션 절반부를 갱신하는 것이다. 다른 정책은 모든 변경을 수집하고, 그것을 링크 또는 네트워크(19)를 통해, 명시적인 요구에 대해서만 단일 콜백으로 다시 애플리케이션(18)으로 전송하는 것이다.

프록시(16)와 위지트(15) 사이에서의 견고성 있고 적절하게 빠른 전송을 위해, 본 발명을 이용하는 시스템은 또한 다음의 특징을 가질 수 있다.

* 프록시(16)와 위지트(15) 사이의 통신은 사용되는 네트워크(19)(또는 링크)의 특성(대역폭, 대기시간)에 적응할 수 있다.

* 위지트(15)의 구현은 실시가능한 한 무상태이며, 따라서, 애플리케이션(18)의 완전한 사용자 인터페이스가 (네트워크 인터럽트 또는 사용자 스테이션/UI 엔진(13/17)의 리셋 이후에) 프록시(16)의 상태로부터 언제라도 재구성될 수 있다.

이상적인 네트워크 상태, 즉, 고대역폭 및 저대기시간의 경우에, 모든 요구 및 데이터는 프록시(16)로부터 관련 위지트(15)로 즉시 전송될 수 있다. 그러나, 작은 네트워크 대역폭의 경우에, 제한된 용량이 최적으로 사용되어야 하며, 즉, 전송이 최소로 감소되어야 한다. 전송되는 데이터의 양을 감소시키기 위해, 실제적으로 가시적인 그런 정보만이 전송된다. 만일 예를 들어, 애플리케이션(18)이 수천개의 데이터 셋트가 제시되어야 하는 테이블을 갖고 있다면, 이들이 모두 초기에 UI 엔진(17)으로 전송되는 것은 아니다. UI 엔진(17)은 가시적인 라인에 대한 데이터 셋트만을 전송할 것을 요구한다. 또한, 아이콘, 이미지, 경보(alerts) 등과 같은 정적인 정보는 한번만 UI 엔진(17)으로 전송되며, 필요할 때마다 사용된다.

전술한 방책들은 저대기시간을 가진 네트워크(19)에 적절하다. 만일 대기시간이 높으면(저속 전송), 다른 방책이 이용된다. 프록시(16)와 위지트(15)는 동기 요구를 피하고 위지트(15)의 국부적 지능을 증가시킴으로써, 가능한 한 분리된다. 테이블 프리젠테이션(이미 전술함)의 경우에, 프록시(16)와 위지트(15) 사이의 모든 요구는 비동기이다. 이것은 위지트가 N개의 데이터 셋트를 요구하는 경우에, 모든 N개의 데이터 셋트가 수신될 때까지 블록킹되지 않는다는 것을 의미한다. 오히려, 요구를 전송한 이후에, 위지트(15)는 일시적인 대리(representative) (위치 홀더) 객체로 국부적인 테이블 구조를 채운다. 요구한 객체는 네트워크 동작에 따라, 소정의 지연 이후에 수신되어, 위치 홀더 객체를 대체한다. 동기 통신을 피하는 것은 또한, 각각의 응답이 수신되기 전에 계류중인(pending) 요구가 다른 처리를 블록킹하는 결과 없이, 많은 작은 요구들을 단일 전송으로 조합할 수 있도록 한다.

통신 요건을 감소시키기 위한 다른 방책

UI 엔진(17)과 애플리케이션(18) 사이의 라운드 트립(round trips)의 수를 감소시키기 위해, 본 발명을 이용하는 시스템은 다음의 공통적으로 사용되는 기능성을 위해 내장 메카니즘을 제공할 수 있다.

(1) 인에이블링 및 디스에이블링

몇몇 위지트(15)상의 특수한 상태는 UI 엔진(17)과 애플리케이션(18) 사이의 어떠한 통신도 없이, 다른 위지트를 인에이블/디스에이블시키기 위해 이용될 수 있다. 그러한 예로는, 엠프티/비-엠프티 필드(empty/non-empty Fields), 리스트 및 테이블 위지트에서의 엠프티/비-엠프티 선택(empty/non-empty selections)이 있다.

(2) 검증(validation)

UI 엔진(17)과 애플리케이션(18) 사이의 어떠한 통신도 없이, 레인지(range) 검사 및 구문 검사 등과 같은 검증을 수행하기 위해 몇몇 위지트(15)에는 사전정의된 밸리데이터(Validator) 및 포맷터(Formatter) 객체가 부가될 수 있다. 이것은 변경이 즉시 전송되지 않고 일괄처리되는 FormModel 기반 위지트에 특히 유용하다.

텍스트 사용자 입력을 처리하고 애플리케이션 프로그램(20)으로부터 수신된 데이터를 발생하는, 즉, 텍스트로 디스플레이되어야 하는 모든 위지트(15)에 대해서는, 데이터 타입에 전형적인 포맷팅 및 검증을 조합한 추상화 데이터타입(abstraction datatype)이 존재한다. 이 객체는 다수의 위지트(15)를 위해 이용될 수 있기 때문에, 단한번만 UI 엔진(17)으로 전송되어야 한다. 이후에, 이 객체는 다른 통신 없이 사용자 입력을 검증하기 위해 이용될 수 있으며, 이것은 반복되는 입력 또는 정정에 대해 질의하고, 출력을 위해 항목을 포맷할 수 있다.

5) 다른 세부사항 및 가능성

(a) 데이터-종속 폼(Data-dependent Form)(페이지북)

특정 상황에서는, 관련 데이터의 값에 따라 사용자에게 상이한 위지트(15)(대화 요소(11))가 도시되어야 한다. 예를 들어, 계정 데이터의 입력을 위해서는, 계정의 형태에 따라 추가적인 입력 필드가 도시되어야 한다. 이것은 사용자 인터페이스가 동적으로 변경되어야 한다는 것을 의미한다. 본 발명을 이용하는 시스템에서, 이러한 "동적인 폼"은 페이지북(Pagebook)을 이용하여 구현될 수 있다. 페이지북은 페이지들의 집합으로 구성되며, 페이지북의 값에 의존하여, UI 엔진(17)은 어떤 페이지가 도시되어야 하는지를 국부적으로 결정할 수 있다. 그러므로, 이러한 경우에 사용자 스테이션(13)과 애플리케이션(18) 사이에

추가적인 전송이 요구되지 않는다.

(b) 복수의 세션

본 발명을 이용하는 시스템에서는, 또한, 사용자 스테이션과 애플리케이션 사이에 복수의 세션을 형성할 가능성이 있다.

1. 단일 사용자 스테이션이 2개 또는 그 이상의 애플리케이션(41,42)과 동시에 작업을 수행할 수 있다. 이 경우에, UI 엔진(40)에서 각각의 세션에 대해 독립된 상이한 위지트 구조가 존재한다. 한편, 각각의 애플리케이션(41,42)은 사용자 스테이션의 UI 엔진(40)(도5 참조)에서 위지트 트리(45,46)중 하나에 대응하는 하나의 프록시 트리(43,44)를 유지한다.

2. 많은 사용자 스테이션들이 동시에 단일 애플리케이션과의 세션을 각각 가질 수도 있다. 그러면, 애플리케이션은 하나의 프록시 구조를 유지하고, 사용자 스테이션내의 각각의 UI 엔진은 하나의 대응하는 위지트 구조를 갖는다.

6) 전송을 위한 세부사항(접속 객체)

사용자 스테이션의 위지트 트리와 애플리케이션에 의해 유지되는 프록시 트리 사이에서의 효율적인 데이터 전송을 위해, 양쪽에 특수한 접속 메카니즘(접속 객체(47))이 제공된다. 도5에는 이들 특수한 접속 메카니즘(47)을 이용하여, 하나의 사용자 스테이션과 2개의 애플리케이션(41,42) 사이에 복수의 세션이 형성되는 경우에 대한 예가 도시되어 있다.

이 경우에, UI 엔진(40)은 다중 스레드(multi-threaded) 구성으로 구현된다. 리스너(listener) 스레드(49)는 애플리케이션(41,42)으로부터의 접속을 수용하고, 각각의 클라이언트에 대한 접속 객체(Connection Object)(47)를 생성한다. 접속 객체(47)의 스레드는 각각의 애플리케이션(41,42)으로부터의 요구를 수신하고, 그것을 사용자 인터페이스 툴킷트의 이벤트 큐(48)로 전달한다. UI 엔진(40)으로부터 애플리케이션(41,42)으로 되돌아오는 콜백 및 이벤트는 관련 접속 객체(47)의 기록 큐(50)로 입력되어, 독립된 라이터(writer)(51) 스레드(도5의 좌측) 내부에서 처리된다.

메인 스레드(52)는 이벤트 큐(48)로부터 이벤트를 판독하여 처리한다. 이들 요구는 나중의 참조를 위해 접속 객체의 레지스트리(57)에 등록되는 객체를 생성하거나, 또는 기존의 객체에 대한 절차를 호출할 수 있다.

위지트 트리(45,46)의 모든 조작은 직렬화되기 때문에, 명시적인 동기화가 요구되지 않는다. UI 엔진(40)으로부터 애플리케이션(41,42)으로의 콜백은 동시에 결과를 대기하지 않기 때문에, 메인 스레드(52)에서는 블록킹이 일어나지 않는다. 이들 설계 결정은 아키텍처를 동적으로 단순화시키고 그 견고성을 개선한다.

애플리케이션측의 아키텍처도 UI 엔진측과 유사하다. 모든 애플리케이션(41,42)은 2개의 스레드(54,55)를 이용하여 비동기 통신을 처리하는 접속 객체(53)를 갖고 있으며, 레지스트리(56)에 프록시를 유지한다(도5의 우측).

복수의 사용자 스테이션이 하나의 애플리케이션과 동시 세션을 갖는 경우에 이와 유사한 구조가 사용된다. 모든 경우에, 위지트 구조 및 프록시 구조 각 쌍에 대해, 대응하는 접속 객체 쌍이 생성된다.

단일 애플리케이션 또는 동일한 데이터베이스(25)(도1)를 이용하는 애플리케이션들과 복수의 세션이 실행되는 경우에, (UI 엔진(40)을 위해 도5의 좌측에 도시된 바와 같은) 직렬화 버퍼(48)가 또한 애플리케이션측에도 필요로 된다. 그렇지 않으면, 동시 억세스로부터 데이터를 보호하기 위해 동기화 메카니즘이 사용되어야 한다.

2개의 스레드(51,58)(송출 및 인입)에 작용하는 이외에도, 접속 객체(47)는 또한, 예를 들어, 위지트 및 프록시 각각에 객체 식별자를 관련시키는 리스트와 같이, 접속과 관련된 모든 자원을 관리한다. 만일 접속이 폐쇄되면, 모든 관련 자원은 해제(released)된다.

7) 애플리케이션 제어기

사용자 스테이션(60)과 애플리케이션(61,62) 사이에 접속을 설정하는 것을 보조하기 위해, 본 발명을 이용하는 시스템에는 애플리케이션 제어기(63)가 제공될 수 있다(도6a 참조). 애플리케이션 제어기(63)는 그 자체가 애플리케이션 프로그램이며, 다른 통상적인 애플리케이션(61,62)이 그것이 실행되고 있는 머신(64) 상에서 개시되고 중단될 수 있도록 하는 프로그램이다. 또한, 이것은 서버(64) 상에서 실행되는 이들 애플리케이션(61,62) 중 어느 것에 대한 접속(66,67)이 설정되도록 요구('connect')하기 위해 그것이 접속되는 UI 엔진(65)에 명령('connectToApp')을 전송할 수 있다. UI 엔진(65)은 하나 이상의 접속(66,67)을 동시에 처리할 수 있다. 이들 접속(66,67)은 수동적으로(passively) 설정될 수 있으며, 그에 따라, UI 엔진(65)은 애플리케이션(61,62)이 접속하기를 서버 모드에서 대기한다. 또한, 그 반대적인 것도 지원되는데, 즉, UI 엔진(65)에 의해 접속(66,67)이 능동적으로 설정될 수 있다. UI 엔진(65)이 시동될 때 명령 라인 인수(argument)에 URL이 명시될 수 있다. 그러나, 특정 접속을 통해 프로그램적으로 요구(애플리케이션의 URL을 포함함)를 전송함으로써 UI 엔진(65)이 애플리케이션 서버(64)에 접속하도록 하는 것도 가능하다.

애플리케이션 제어기(63)는 다수의 애플리케이션을 윈도우(70)(도6b)에 시각적으로 제시한다. 애플리케이션의 표기에 더블 클릭하거나(Dossier 애플리케이션은 필드(72)에 클릭함으로써 개시됨) 또는 애플리케이션을 선택하고 "Connect" 버튼(71)에 클릭함으로써, UI 엔진(65)이 애플리케이션의 서버(64)에 접속하게 된다. 이러한 접속은 다시 애플리케이션을 선택하고 "Disconnect" 버튼(73)에 클릭함으로써 폐쇄될 수 있다. 애플리케이션의 서버(64)에 접속이 설정될 수 있으면, 칼럼(74)내의 적색광을 녹색으로 전환시킴으로써, 대응하는 시각적 표기가 이 상태를 표시할 수 있다. 접속은 또한 계류중일 수도 있는데, 즉, 서버(64)가 응답하지 않음으로 인해, 접속이 UI 엔진(65)에서는 개시되었지만, 아직 설정되지는 않을 수도 있다. 계류중인 접속은 칼럼(74)에서 황색광으로 표시될 수 있다. 애플리케이션의 상태를 디스플레이

또는 표현하기 위한 다른 여러 가지 방법이 있다는 것은 자명하다.

8) 예

이 섹션에서는 2개의 예가 제공되는데, 하나는 테이블 데이터의 전송에 대한 것이고, 다른 하나는 파이 차트(pie chart) 디스플레이를 위한 위지트를 생성하는 것에 대한 것이다.

(a) 테이블 예

본 발명을 이용하는 시스템에서, 프록시(애플리케이션 서버에 제공되는 카운터파트 요소)는 가능한 한 호스트 프로그래밍 언어(예, Java, Smalltalk)를 이용하여 제공된 대화 요소와 유사한 애플리케이션 프로그래밍 인터페이스를 갖는다. 위지트/프록시 구성을 이용하는 경우에 제공되는 다양한 액션은 프록시의 인터페이스의 통신 특성에 관한 매우 상이한 요건을 가질 수 있다. 그러므로, 사용자는 위지트/프록시를 이용하는 경우에 통신 상태에 강력하게 적응해야 한다.

그러나, 본 발명을 이용하는 시스템에서는, 통신 프로토콜이 애플리케이션 프로그래밍 인터페이스로부터 대부분 분리된다. 다음에, 이것은 테이블 모델의 경우에 대해 도시되게 된다(도7 참조).

UI 엔진(80)내의 테이블 모델은 다음 명령을 주로 수용한다.

* changed:이것은 전체 새도우 테이블 모델(또는 그 지정된 부분)을 무효로 셋트하도록 UI 엔진(80) 내의 모델에 통지한다. 만일 이것이 시각적 프리젠테이션에 작용하면, 명령 requestData에 의해 새로운 데이터가 요구된다.

* sendData:이것은 UI 엔진(80) 내의 새도우 테이블 모델로 데이터(한 블록이 다수의 라인 및/또는 칼럼을 포함함)를 전송한다. 이러한 전송은 명령 requestData에 대한 응답으로서 수행될 수 있으며, 또는 자발적으로 수행될 수도 있다.

애플리케이션(81)측의 프록시는 다음의 요구를 처리한다.

* requestData:이것은 애플리케이션(81)으로부터 데이터 블록을 요구한다. 호출은 비동기로 이루어지며, 즉, UI 엔진(80)을 블록킹하지 않고, 그러므로, 데이터의 즉시 전달을 초래하지 않는다. 그러나, 데이터가 응답가능한 시간주기 내에서 전송될 것으로 예상한다.

* setData:이것은 UI 엔진(80)으로부터 애플리케이션(81)으로 사용자에 의해 변경된 테이블 데이터를 전송한다.

다음에는, 도7을 참조하여 UI 엔진(80)과 애플리케이션(81) 사이에서의 데이터 교환의 통상적인 예에 관해 설명한다. 제1 단계(82)에서, 애플리케이션(81)은 테이블의 크기에 관해 UI 엔진(80) 내의 모델에 통지한다. 본 예에서, UI 엔진(80)은 테이블의 첫 번째 10개의 라인을 디스플레이해야 한다. 통신 특성으로 인해, 명령 requestData(0, 19)을 발생함으로써 UI 엔진(80)에 의해 20개의 라인을 요구받는다(단계 83). 이제, 애플리케이션(81)은 데이터를 전송하지만, 단지 첫 번째 10개의 라인만을 전송할 것을 결정한다(단계 84). 이에 관한 명령은 sendData(0, 10, Data) 이다. UI 엔진(80)이 요구한 나머지 10개의 라인을 전송하기 이전에, 애플리케이션(81)은 전체 테이블을 무효로 만든다(단계 85). 그리고, 명령 changed(ROWS, 0, 999)을 전송함으로써, 전체 새도우 테이블 모델을 무효로 셋트하도록 UI 엔진(80) 내의 모델에 통지한다. 그러면, UI 엔진(80)은 다시 테이블의 가시 영역에 대한 데이터만을 요구한다. 본 예에서는 첫 번째 20개의 라인이 가시적이기 때문에, UI 엔진(80)은 명령 requestData(0, 19)을 발생한다(단계 86). 마지막 단계(87)에서, 애플리케이션(81)은 이제 요구한 데이터를 UI 엔진(80)으로 전송한다. 이에 관한 명령은 sendData(0, 20, Data)이다.

도7에 도시된 방식을 구현할 때, 통신 특성이 고려되게 된다. 먼저, 애플리케이션(81) 측에서의 테이블 데이터의 변경이 change(변경) 요구를 전송하는 것을 초래하는지 또는 변경된 데이터가 명령 sendData를 이용하여 UI 엔진(80)으로 직접 전달될 것인지에 관한 선택이 이루어진다. 첫 번째 솔루션은 통신 노력을 증가시키만, 전송되는 데이터의 양을 최소로 감소시킨다. 두 번째 솔루션은 통신 노력을 최소화시키지만, 가시적인 아닌 데이터(디스플레이를 위해 결코 요구되지 않을 수도 있음)를 전송할 가능성이 있다.

b) 파이 차트 예

다음에는 도8의 파이 차트(pie chart)의 예를 참조하여 위지트의 생성에 관해 설명한다.

전술한 바와 같이, 대화 요소에 대한 위지트/프록시 구성은 2개의 절반부, 즉, UI 엔진에 있는 프리젠테이션 요소(위지트)와 애플리케이션 서버에 있는 카운터파트 요소(프록시)로 이루어진다. 이들 요소(절반-객체) 뿐만 아니라, 종종 이미 존재하고 본 발명을 이용하는 시스템을 위해 적응될 필요가 없는 리얼(real) 대화 요소(80)가 있다. 리얼 대화 요소와 UI 엔진내의 프리젠테이션 요소(위지트)는 Java로 구현된다. 애플리케이션 서버내의 카운터파트 요소(프록시)는 이 시스템에 의해 지원되는 언어(예, Smalltalk, Java)로 구현되어야 한다.

파이 차트 구현 세부사항

파이 차트(90)는 도시될 값, 그 명칭 및 칼라를 셋팅하는 것을 지원한다. 파이 세그먼트(91-93)에 클릭함으로써, 클릭된 세그먼트의 명칭을 가진 액션이 인수(argument)로서 전송된다.

간략성을 위해, 파이 차트(90)는 모델-기반 위지트로서 구현되지 않는다. 모델-기반 위지트를 구현하는 것은 모델 객체를 억세스하고 그 변경을 추적하는 대응하는 절반 객체의 구현을 필요로 하게 된다.

위지트(프리젠테이션 요소)의 구현

UI 엔진이 사례생성(instantiate)을 필요로 하는 경우에, 그것은 프록시(16)에 대응하는 위지트(15)를 찾기 위해 상이한 룹을 이용한다. 이 경우에, 2개의 절반 객체를 연관시키는 것은 위지트의 명칭을 기반

으로 한다. 관례는 UI 엔진 위지트에 대한 UI 엔진 프리픽스(prefix)(UIPieChart)와 프록시 절반부에 대한 프리픽스(ULCPieChart)를 이용하는 것이다. 각각의 UI 엔진 위지트는 프록시로부터 그 상태를 초기화해야 한다. 또한, 파이차트(90)는 다음의 애플리케이션 프로그래밍 인터페이스를 제공한다.

1. 데이터(값, 레이블, 칼라)를 셋트하기 위한 요구를 처리한다.

2. 파이 세그먼트가 클릭되는 경우에 프록시(16)로 이벤트를 전송한다.

UIPieChart는 com.oti.ulc.Component로부터 강하된다. UIPieChart는 파이차트가 UI 엔진에 의해 이용될 수 있도록 그것을 적응시킨다. 이렇게 하기 위해, 파이차트 위지트에 대한 레퍼런스를 유지한다. 다음은 UIPieChart의 클래스 정의로부터의 인용구이다.

```
-----------------------------------------------------
public class UIPieChart extends UIComponent implements
ActionListener {
    private PieChart fPieChart= null;
}
-----------------------------------------------------
```

프록시로부터 그 상태를 복원하기 위해, UIPieChart는 restoreState(상태복원)을 오버라이드한다(overrides).

```
-----------------------------------------------------
public void restoreState(ORBConnection conn, Anything args) {
    super.restoreState(conn, args);
    fPieChart= new PieChart(args.get("w", 200), args.get("h",
150));
    fPieChart.setData(args.get("data"));
    fPieChart.addActionListener(this);
}
-----------------------------------------------------
```

계승된(inherited) restoreState 호출 절차는 계승된 상태를 복원한다. 파이차트 위지트는 복원된 상태를 이용하여 생성되고 초기화된다. ORBConnection 인수는 이 예에서는 사용되지 않으며, 단지 베이스 클래스로 전달된다.

UI 엔진과 프록시 사이의 데이터 통신은 Anythings로 불리는 데이터 객체에 근거하여 이루어진다. Anything은 프로세스들 사이에서 전송될 수 있는 동적인 데이터 구조이다. 절차 restoreState는 그 인수로서 Anything을 수신하고, 개개의 인수를 검색하기 위해 Anything 억세서(accessor) 절차를 이용한다. Anything은 간단한 데이터 형태를 포함하거나 또는 Anything의 어레이 및 사전(dictionaries)을 포함할 수 있다. 인수를 검색하는 것은 프록시 및 UI 엔진 위지트가 그들이 그의 인수들을 Anything으로 어떻게 패키징할 것인가에 관해 합치하는 것을 필요로 한다. 이 경우에, Anything은 사전이 되며, restoreState는 명칭에 의해 인수를 검색한다. 예를 들어, args.get("w")는 파이 차트의 폭 인수(width argument)를 검색한다. 동일한 방식으로, 절차 setData는 Anything으로부터 파이 차트의 데이터를 복원하고, 그것을 PieChart 위지트로 전달한다.

UIPieChart는 파이 세그먼트가 클릭될 때 프록시에 통지하길 원하기 때문에, 그것은 ActionListener(액션 리스너) 인터페이스를 구현한다. 이것은 또한, 그 자체를 PieChart의 액션 리스너로서 등록한다. 이 절차는 Anything으로서 패키징된 그 인수와 함께 그 요구의 명칭을 수신한다. UIPieChart는 파이 차트의 데이터를 셋트하기 위해 setData로 명명된 단일 요구만을 구현한다.

```
-----------------------------------------------------
public void handleRequest(ORBConnection conn, String request,
Anything args){
    if (request.equals("setData")) {
        setData(args);
        return;
    }
    super.handleRequest(conn, request, args);
}
-----------------------------------------------------
```

절차 handleRequest는 Anything으로부터 데이터를 추출하고, 그것을 PieChart내로 설치하기 위해 절차 setData를 이용한다. 계승된 handleRequest는 베이스 클래스가 그 요구를 처리할 수 있도록 한다.

UIPieChart는 사용자가 파이 세그먼트를 클릭할 때 프록시로 이벤트를 전송할 필요가 있다. 이렇게 하기 위해, 그것은 액션 리스너 절차 actionPerformed를 구현하고, 이벤트를 전송하기 위해 sendEventULC를

호출한다.

```
public void actionPerformed(ActionEvent e) {
    sendEventULC("action", "cmd", new
Anything(e.getActionCommand()));
}
```

sendEventULC는 이벤트 명칭, 그 형태명 및 인수를 취한다. 이벤트의 인수는 Anything으로 래핑(wrap)된다. 이 경우에, 이것은 클릭된 파이 세그먼트의 레이블에 대응하는 간단한 스트링이 된다.

프록시(카운터파트 요소)의 구현

전술한 예는 UI 엔진 절반부에서 요구를 어떻게 처리하고 이벤트를 전송할 것인가 보여 주었다. 이 섹션은 파이차트(90)의 프록시 절반부를 어떻게 구현할 것인가에 관해 설명한다. PieChart 위지트는 Java 및 Smalltalk 둘다로부터 이용가능할 필요가 있기 때문에, 두 언어 모두에 대한 구현이 존재한다.

Java로의 프록시 구현

PieChart의 자바 절반부는 클래스 ULCPieChart에서 정의된다. UIPieChart와는 대조적으로, 이것은 선택된 패키지로 ULCPieChart를 정의할 수 있다. 이러한 구현은 ULC를 이용하여 프록시에 접두사를 붙이는 규칙을 이용한다. 내부적으로, 프록시가 그 UI 엔진 위지트를 생성하길 원할 때, 그것은 생성될 위지트의 형태명을 UI 엔진으로 전달한다. 디폴트에 의해, 형태명은 ULC 프리픽스 없이 클래스명에 대응한다. 원하는 경우에, 이 디폴트는 ULCProxy에 정의된 절차 TypeString을 오버라이드함으로써 변경될 수 있다.

프록시는 대응하는 UI 엔진 절반부의 상태를 유지해야 한다. 이렇게 하기 위해, ULCPieChart는 파이차트 위지트의 값, 레이블 및 칼라를 그 사례 변수에 저장한다.

```
public class ULCPieChart extends ULCComponent {
    protected double[] fValues;
    protected String[] fColors;
    protected String[] fLabels;
    int fWidth;
    int fHeight;
}
```

ULCPieChart는 그 프록시 클래스의 위치를 UI 엔진에 알려줄 필요가 있다. 이것은 전적으로 적임의 클래스 경로를 응답하는 typeString 절차를 구현함으로써 수행된다.

```
public String typeString() {
    return "ULCExtensions.UIPieChart";
}
```

ULCPieChart는 UI 엔진 요구를 대칭적으로 구현해야 한다. 먼저, 이것은 위지트 상태를 UI 엔진으로 전송할 필요가 있다. 이것은 saveState를 오버라이드함으로써 수행된다. saveState는 Anything 내로 그 사례 변수에서 유지되는 인수를 채운다.

```
public void saveState(Anything a) {
    super.saveState(a);
    a.put("w", fWidth);
    a.put("h", fHeight);
    Anything data= new Anything();
    fillData(data);
    a.put("data", data);
}
```

setData 요구는 다음의 ULCPieChart 절차로 구현된다.

```
public void setData(String[] labels, double[] values, String[]
colors) {
    fValues= new double[values.length];
    fColors= new String[colors.length];
    fLabels= new String[labels.length];
    System.arraycopy(labels, 0, fLabels, 0, labels.length);
    System.arraycopy(values, 0, fValues, 0, values.length);
    System.arraycopy(colors, 0, fColors, 0, colors.length);
    Anything data= new Anything();

    fillData(data);
    sendUI("setData", data);
```

요구 절차는 단지 그 인수를 Anything내로 패키징할 필요가 있다. 이 경우에, 값, 레이블 및 칼라를 가진 어레이가 UI 엔진 절반부가 예상하고 있는 Anything 구조 내로 래핑(wrap)된다. 다음에, 이 요구는 sendUI를 이용하여 UI 엔진으로 전송되고, 이것은 요구명과 Anything 인수를 UI 엔진으로 전송한다.

마지막 단계는 UI 파이차트(90)로부터 액션 이벤트를 처리하는 것이다. 요구를 처리하기 위해, 이것은 handleRequest를 오버라이드한다. 이 절차는 요구 데이터를 저장하고 있는 Anything과 함께 요구명을 수신한다.

```
public void handleRequest(ORBConnection conn, String request,
Anything args) {
   if (request.equals("event")) {   // (1)
       String type= args.get("type", "???");
       if (type.equals("action")) // (2)
           distributeToListeners(new ULCActionEvent(this,
args.get("cmd", "???"))); // (3)
       return;
    }
    super.handleRequest(conn, request, args);
}
```

다음은 handleRequest에서 구현될 단계들이다.

1. 요구가 이벤트인지 검사한다.

2. 그것이 액션 이벤트인지 알기 위해 이벤트 요구의 형태를 검사한다.

3. 그것이 액션 이벤트인 경우에, ULCActionEvent를 생성하고, 등록된 리스너에 통지하기 위해 distributeToListner(리스너로의 분배) 절차를 이용한다.

4. 그것이 액션 이벤트가 아닌 경우에는 그것을 베이스 클래스로 전달한다.

Smalltalk로의 프록시 구현

ULCPieChart의 스몰토크 구현은 개념적으로는 자바 구현과 동일하다. saveState는 위지트 데이터를

Anything의 스몰토크 버전으로 패킹(pack)하는 절차이다.

```
saveState: aStcAnything

 super saveState: aStcAnything.
 aStcAnything
   at: 'w' put: self width;
   at: 'h' put: self height;
   at: 'data' put: (self
                    fillData: self values
                    colors: self colors
                    labels: self labels);
 ycurself
```

다음은 setData 요구의 구현예이다.

```
setData: aValuesCollection colors: aColorsCollection labels:
aLabelsCollection
    | data |

    data := self
              fillData: aValuesCollection
              colors: aColorsCollection
              labels: aLabelsCollection.
    self sendTcUI: 'setData' with: data
```

스몰토크에서, UI 엔진 위지트의 생성을 요구하기 위해 이용되는 형태명은 절차 typeString을 오버라이드함으로써 정의된다.

```
typeString
    ^'ULCExtensions.PieChart'
```

ULCPieChart의 경우에, 단순히 스트링 'PieChart'를 응답한다.

발명의 효과

본 발명은 대화 세션에서의 데이터 전송을 위한 프리젠테이션 로직의 일부분이 사용자 스테이션에 구현될 수 있도록 하며, 그럼에도 불구하고, 다양한 애플리케이션에 보편적으로 적절한 사용자 스테이션을 구비할 수 있도록 한다. 사용자 스테이션은 새로운 애플리케이션이나 변경된 애플리케이션이 사용되는 경우에도 수정될 필요가 없다. 단지 대화 세션의 개시점에서 몇몇 인터페이스 요소 구조(위지트 구조(widget structure))를 정의할 필요가 있다. 링크의 양쪽에서의 동일한 데이터의 중간 휴딩으로 인해, 전송이 사용자 스테이션과 애플리케이션에 투명해진다. 독립적인 전송과 어느 한쪽에서의 대응하는 로직 인터페이스 요소에서 다른 필요한 데이터를 준비하면서, 실제적으로 요구되는(가시적인) 데이터만을 전송할 가능성으로 인해, 대화식 협동이 링크 및 네트워크 상태에 상당한 정도로 독립적이 된다.

(57) 청구의 범위

청구항 1

링크 또는 네트워크를 통해 서버내의 애플리케이션 프로그램과 사용자 스테이션 사이에서의 대화식 협동을 위한 시스템에 있어서,

상기 애플리케이션 프로그램과 상기 사용자 스테이션 사이에 제공되는 프리젠테이션 계층; 및

상기 사용자 스테이션 상에서의 정보의 프리젠테이션을 위해 사용되는 대화 요소

를 포함하고,

상기 시스템에서,

상기 프리젠테이션 계층은 부분적으로는 상기 사용자 스테이션에서 구현되고, 부분적으로는 상기 서버에서 구현되며, 상기 대화 요소를 표현하는 한쌍의 프리젠테이션 요소를 포함하고, 상기 쌍중에서 하나의 프리젠테이션 요소는 사용자 스테이션에 제공되고, 상기 쌍중에서 대응하는 프리젠테이션 요소는 서버에 제공되며:

상기 사용자 스테이션 내의 프리젠테이션 요소를 독립적으로 제어하기 위한 프로그램 객체가 상기 사용자 스테이션에 제공되는

대화식 협동 시스템.

청구항 2

사용자 스테이션과 애플리케이션 프로그램을 구비한 서버, 및 상기 사용자 스테이션과 상기 서버 사이에서의 데이터 교환을 위한 링크 또는 네트워크를 포함하는 시스템에 있어서,

상기 사용자 스테이션과 상기 애플리케이션 프로그램 사이에서 대화 데이터 및 메시지의 대화식 전송을 위해 제공되는 프리젠테이션 인터페이스

를 포함하고,

상기 시스템에서,

프리젠테이션 계층은 상기 사용자 스테이션과 상기 애플리케이션 프로그램 사이에 제공되고, 사용자 스테이션에서 대화 데이터의 프리젠테이션 및/또는 입력을 위해 사용되는 대화 요소를 위해, 대화 데이터를 전송하고 유지하기 위한 한쌍의 인터페이스 요소 - 상기 한쌍중 하나는 사용자 스테이션에 위치하고, 다른 하나는 서버에 위치함 - 를 제공하며;

상기 사용자 스테이션에 위치한 인터페이스 요소를 제어하기 위한 프로그램 객체가 상기 사용자 스테이션에 제공되고,

상기 사용자 스테이션과 상기 서버에서 각각 상기 한쌍의 인터페이스 요소의 대응하는 인터페이스 요소 사이에서 대화 데이터를 독립적으로 교환하기 위한 메카니즘이 제공되는

시스템.

청구항 3

사용자 스테이션 상에서 대화 데이터의 입력 및/또는 출력을 위한 대화 요소를 이용하여, 링크 또는 네트워크를 통해 애플리케이션 프로그램과 원격 사용자 스테이션의 대화식 협동을 위한 방법에 있어서,

상기 사용자 스테이션에서, 상기 대화 데이터를 전송하고 유지하기 위한 프리젠테이션 요소의 구조와 상기 프리젠테이션 요소를 구현하고 제어하기 위한 관리 메카니즘을 설정하는 단계;

상기 애플리케이션 프로그램을 이용하여, 상기 대화 데이터를 전송하고 유지하기 위한 카운터파트 요소의 대응하는 구조를 설정하는 단계; 및

상기 대화 데이터의 대화식 교환을 위해, 관련 프리젠테이션 요소 및 대응하는 카운터파트 요소에서 각각 상기 대화 요소를 위한 2개의 대응하는 데이터 셋트를 유지하는 단계

를 포함하고,

따라서, 상기 사용자 스테이션과 상기 애플리케이션 프로그램은 각각, 링크 또는 네트워크를 통한 전송과 무관하게 상기 프리젠테이션 요소 및 카운터파트 요소와 협동할 수 있는

대화식 협동 방법.

청구항 4

제 1 항 또는 제 2 항에 있어서,

상기 사용자 스테이션내의 인터페이스 요소와 상기 애플리케이션 프로그램에 의해 제공된 인터페이스 요소에 대해, 데이터 구조 모델이 제공되고, 이들 양쪽의 대응하는 모델은 대칭적이며, 따라서, 두 모델 모두의 각각의 위치에 동일한 데이터가 저장될 수 있는

시스템.

청구항 5

제 4 항에 있어서,

상기 모델은 상기 사용자 스테이션내의 관련 인터페이스 요소 및 상기 서버에 의한 인터페이스 요소로부터 분리되어 제공되며, 따라서, 양쪽 모두의 대응하는 모델들 사이에서 데이터가 직접 전송될 수 있는

시스템.

청구항 6

제 4 항에 있어서,

동일한 데이터 구조로부터 상이한 데이터 셋트를 필요로 하는, 상기 서버에 의해 제공된 다수의 프리젠테이션 요소에 대해, 공통 모델이 제공되는

시스템.

청구항 7

제 1 항 또는 제 2 항에 있어서,

상기 서버에 의해 제공된 상기 인터페이스 요소와 상기 사용자 스테이션에 제공된 상기 인터페이스 요소

는 그들이 사용되는 애플리케이션에 따라 트리 구조로 구성되고, 상기 사용자 스테이션 내의 구조는 상기 애플리케이션 프로그램으로부터의 명령에 따라 상기 사용자 스테이션에 제공된 프로그램 객체에 의해 구현되는

시스템.

청구항 8

제 1 항 또는 제 2 항에 있어서,

상기 사용자 스테이션 내의 프리젠테이션 요소들의 각각의 구조와 상기 서버에 의해 제공된 인터페이스 요소들의 각각의 구조에 대해, 통신 메카니즘이 제공되고, 상기 각각의 통신 메카니즘은 양방향 데이터 전송을 위한 링크 또는 네트워크에 접속되고, 상기 인터페이스 요소들의 관련 구조로/로부터 데이터 또는 요구의 전송을 위해 버퍼링 기능을 가진 분리된 입력 및 출력 경로들을 구비한

시스템.

청구항 9

제 8 항에 있어서,

상기 통신 메카니즘의 전송 특성은 상기 링크 또는 네트워크의 현재의 전송 특성에 적응하도록 선택적으로 수정될 수 있는

시스템.

청구항 10

제 1 항 또는 제 2 항에 있어서,

상기 사용자 스테이션 상의 정보의 프리젠테이션은 대화 데이터 및/또는 메시지의 입력 및/또는 출력인

시스템.

청구항 11

제 1 항 또는 제 2 항에 있어서,

상기 한쌍의 프리젠테이션 요소는 데이터, 바람직하게는 대화 데이터의 교환을 위해 사용되는

시스템.

청구항 12

제 1 항 또는 제 2 항에 있어서,

상기 한쌍 중에서 상기 사용자 스테이션 제공되는 인터페이스 요소는 위지트 또는 프리젠테이션 요소이고, 상기 한쌍 중에서 상기 서버에 의해 제공되는 대응하는 프리젠테이션 요소는 프록시 또는 카운터파트 요소인

시스템.

청구항 13

제 1 항 또는 제 2 항에 있어서,

상기 프로그램 객체는 사용자 인터페이스 엔진인

시스템.

청구항 14

제 3 항에 있어서,

대화 세션을 위해 상기 애플리케이션 프로그램에 의해 상기 카운터파트 요소에 로드된 데이터 셋트로부터, 상기 사용자 스테이션에 의해 실제적으로 필요로 되는 데이터만이 상기 프리젠테이션 요소로 전송되는

방법.

청구항 15

제 3 항에 있어서,

상기 애플리케이션으로부터 상기 사용자 스테이션으로의 대화 데이터의 비동기 전송을 위해, 상기 프리젠테이션 요소는 상기 사용자 스테이션에서 로딩 프로세스를 나타내기 위한 위치 홀더 항목으로 채워지고, 상기 위치 홀더 항목은 각각의 카운터파트 요소로부터 링크 또는 네트워크를 통해 수신되는 실제 데이터 항목에 의해 대체되는

방법.

청구항 16

제 3 항에 있어서,

상기 서버에 의해 제공된 카운터파트 요소에 포함된 데이터를 이용하여, 상기 사용자 스테이션의 프리젠테이션 요소에 필요한 데이터를 재로딩함으로써, 인터럽트 이후에 대화 세션을 재개시하는 단계

를 더 포함하는 방법.

청구항 17

제 3 항에 있어서,

링크 또는 네트워크를 통해 상기 애플리케이션 프로그램으로부터 인에이블링 또는 디스에이블링 정보를 전송하는 것을 피하기 위해, 적어도 하나의 다른 국부적 프리젠테이션 요소의 상태에 의존하여 상기 사용자 스테이션 내의 프리젠테이션 요소를 인에이블 또는 디스에이블시키는 단계

를 더 포함하는 방법.

청구항 18

제 3 항에 있어서,

상기 프리젠테이션 요소와 상기 카운터파트 요소를 트리 구조로 구성하고, 대화 세션의 개시정에서 상기 애플리케이션 프로그램의 제어하에 상기 사용자 스테이션에서 트리 구조를 초기화하는 단계

를 더 포함하는 방법.

청구항 19

제 18 항에 있어서,

현재 상태를 갱신하기 위해, 대화 세션 동안에 상기 애플리케이션 프로그램의 제어하에 상기 사용자 스테이션 내의 프리젠테이션 요소의 트리 구조를 수정하는 단계

를 더 포함하는 방법.

청구항 20

대화식 시스템에서, 애플리케이션 프로그램을 호스팅하는 설비에 네트워크 또는 링크를 통해 접속된 사용자 스테이션에서의 데이터의 프리젠테이션을 위한 방법에 있어서,

사용자 스테이션에서, 상기 애플리케이션 프로그램으로부터 데이터를 수신하고 유지하기 위한 프리젠테이션 요소를 설정하는 단계;

상기 사용자 스테이션에서, 상기 프리젠테이션 요소를 제어하기 위한 관리 메카니즘을 설정하는 단계;

상기 설비에서, 데이터를 전송하고 유지하기 위한 카운터파트 요소의 대응하는 구조를 설정하는 단계; 및

상기 설비와 상기 사용자 스테이션 사이에서의 데이터의 대화식 교환을 위해, 상기 프리젠테이션 요소와 상기 대응하는 카운터파트 요소에서 2개의 대응하는 데이터 셋트를 유지하는 단계

를 포함하고,

따라서, 상기 사용자 스테이션이 사용자로의 프리젠테이션을 위해 상기 프리젠테이션 요소에서 유지되는 데이터를 이용할 수 있는

데이터 프리젠테이션 방법.

청구항 21

제 20 항에 있어서,

상기 대화식 시스템은 보험업, 은행업, 제조업, 호텔업, 또는 콜센터 애플리케이션인

데이터 프리젠테이션 방법.

청구항 22

제 21 항에 있어서,

상기 설비는 상기 애플리케이션 프로그램에 데이터를 제공하는 데이터베이스 또는 비지니스 서비스에 접속되고, 상기 데이터는 보험업, 은행업, 제조업, 호텔업 또는 콜센터 특유의 데이터인

데이터 프리젠테이션 방법.

청구항 23

제 20 항에 있어서,

상기 사용자 스테이션으로 새로운 데이터를 입력하는 단계; 및

상기 프리젠테이션 요소로부터 상기 새로운 데이터를 상기 링크 또는 네트워크 및 상기 대응하는 카운터파트 요소를 통해 상기 애플리케이션 프로그램으로 전송하는 단계

를 더 포함하는 데이터 프리젠테이션 방법.

청구항 24

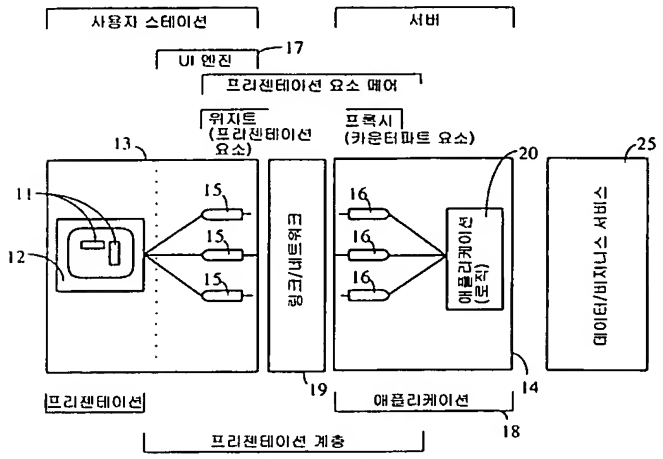
제 23 항에 있어서,

상기 프리젠테이션 요소로부터 상기 새로운 데이터를 상기 링크 또는 네트워크 및 상기 대응하는 카운터파트 요소를 통해 상기 애플리케이션 프로그램으로 전송하기 전에, 상기 사용자 스테이션 내에서 국부적으로 상기 새로운 데이터를 검증하는 단계

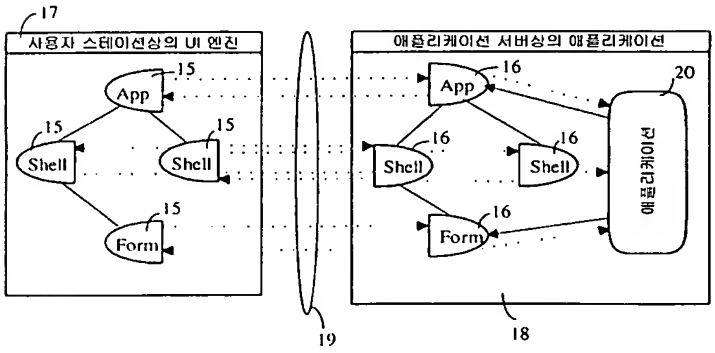
를 더 포함하는 데이터 프리젠테이션 방법.

도면

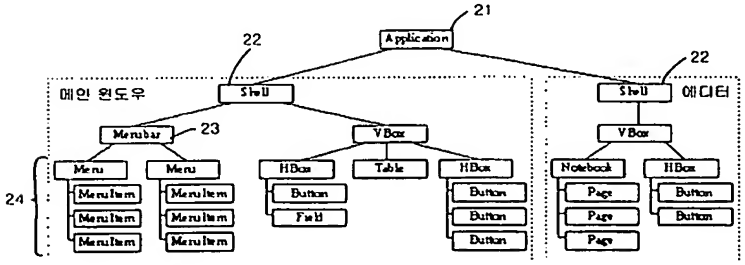
도면1



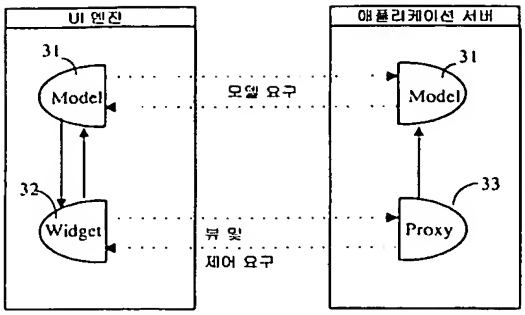
도면2



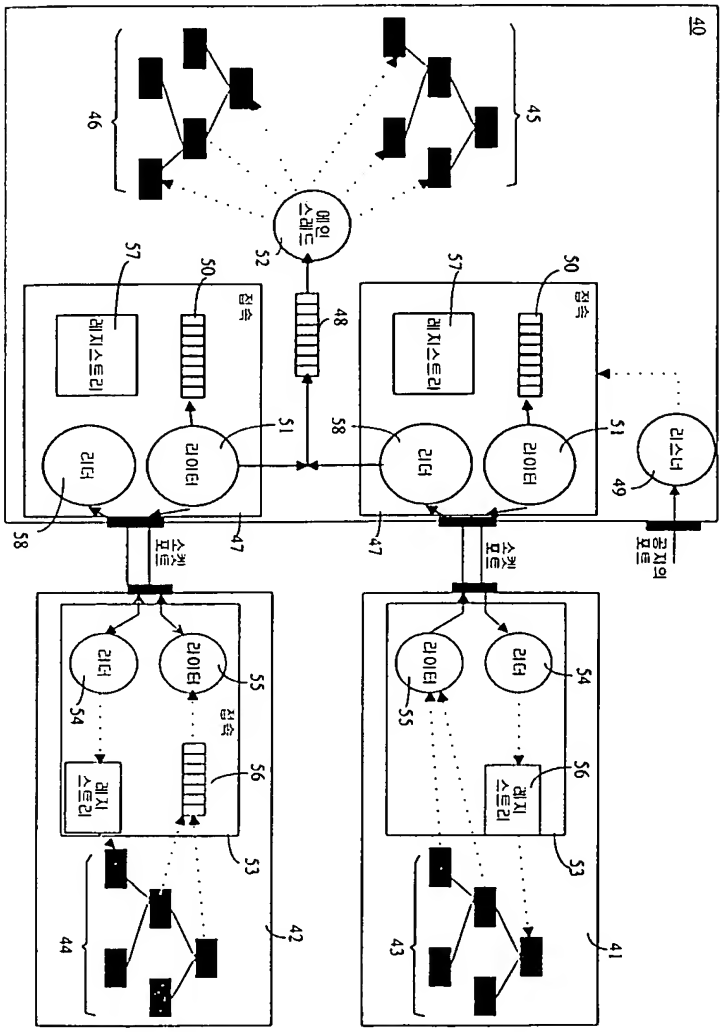
도면3



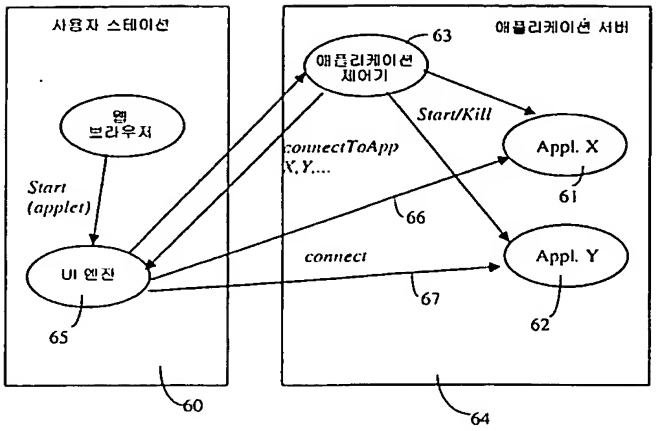
도면4


UI 엔진
애플리케이션 서버
31
Model
모델 요구
31
Model
32
Widget
33
Proxy
뷰 및
제어 요구

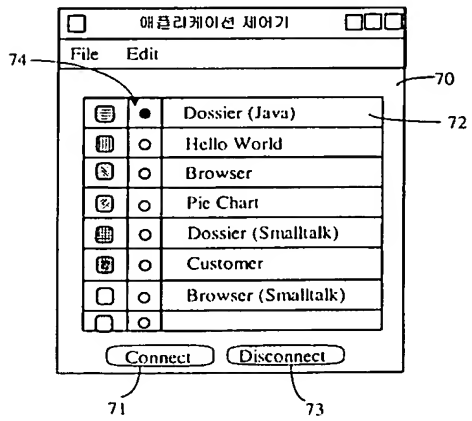
도면5


40
46
45
메인
스레드
52
57
50
레지스트리
접속
48
51
58
리더
라이터
49
리스너
47
소켓
포트
원격
포트
54
55
56
53
42
41
44
43

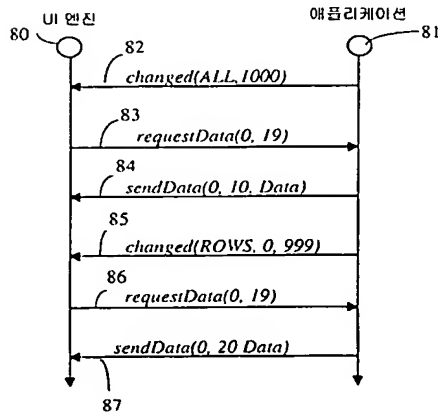
도면6a



도면6b



도면7



도면8